EXERCISES QUE TRABAJAREMOS EN LA CUE:

- EXERCISE 1: SELECTORES BÁSICOS
- EXERCISE 2: MANIPULANDO LOS VALORES DEL DOM
- EXERCISE 3: CAPTURANDO Y MODIFICANDO UN VALOR
- EXERCISE 4: CONOCIENDO LOS EVENTOS DE JAVASCRIPT

EXERCISE 1: SELECTORES BÁSICOS

Todo comienza con un proceso conocido como *Critical Rendering Path*, en el cual los navegadores convierten en pixeles aquel código que hemos escrito en nuestros archivos **JavaScript**, **HTML** y **CSS**. En este paso el navegador crea dos Arboles:

- **DOM**: representación del HTML. Su estructura en forma de árbol contiene nodos, además, es dinámico.
- **CSSOM**: Conjunto de APIs que permite manipular CSS desde JavaScript. Permite leer y modificar el estilo de CSS de forma dinámica.

El **DOM** este compuesto por nodos, donde cada nodo son etiquetas **HTML** y el **CSSOM** hace su parte con la manipulación de estilos.

LEER NODOS

Vamos a generar un **HTML** simple, que se encuentre ligado a un archivo de tipo **JavaScript** para poder manipular estos elementos:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-
scale=1.0">
  <title>Document</title>
</head>
```

```
 9 <body>
10
11    <div class="contenedor">
12       <p id="parrafo">Hola soy un párrafo.</p>
13    </div>
14
15    <div class="contenedor">
16       <h1 >Soy un título</h1>
17    </div>
18
19
20
21 <script src="index.js"></script>
22 </body>
23 </html>
```

- **Document.getElementById**: Nos permite obtener un elemento del **DOM** a través de su **ID**. Para este caso, el nodo retornado del **HTML** es único, ya que el **ID** debiese ser único.

```
1 var selectorId = document.getElementById('parrafo');
```

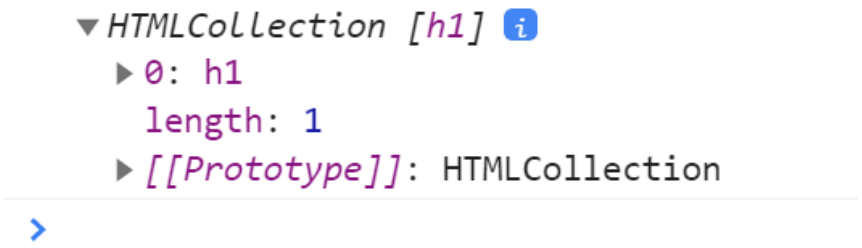
Obtenemos como resultado:

```
    <p id="parrafo">Hola soy un párrafo.</p>
>
```

- **Document.getElementsByTagName**: nos permite obtener una lista con todos los nodos que tienen definida la etiqueta proporcionada. Puede obtener, 0, 1 o N nodos.

```
1 var selectorEtiqueta = document.getElementsByTagName("h1");
```

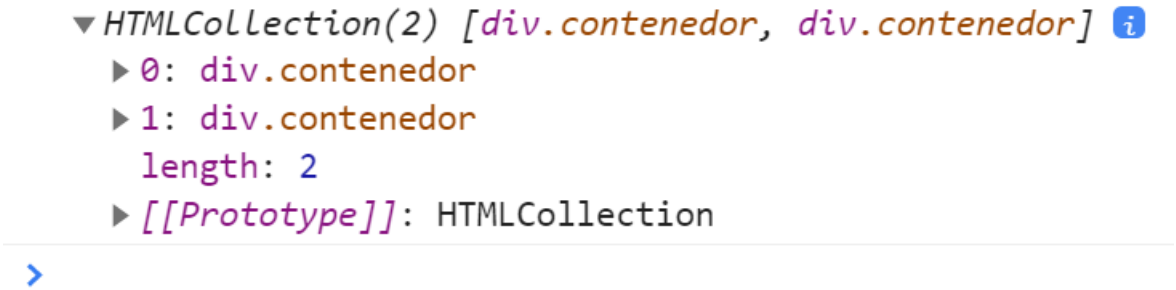
Obtenemos como resultado:



- **Document.getElementsByClassName**: retorna una lista de una clase especifica. Es decir, traemos todos los elementos que contengan dicha clase.

```
var selectorClass = document.getElementsByClassName("contenedor");
```

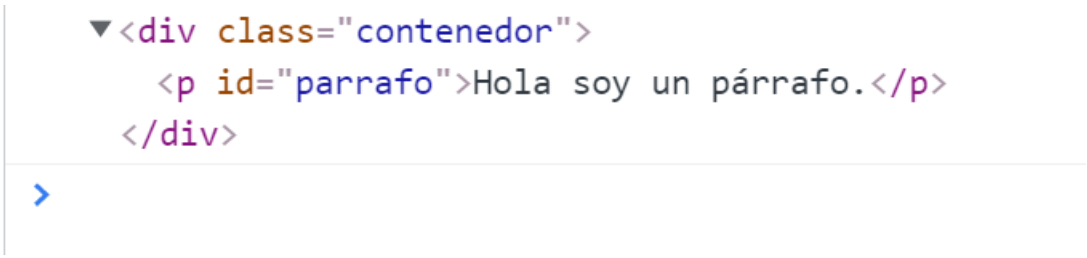
Obtenemos como resultado:



- **Document.querySelector**: nos permite obtener cualesquiera elementos del **DOM** de acuerdo con el argumento que le indiquemos. Si varios elementos coinciden con la búsqueda, retornará el primero que encuentre. Si no encuentra ningún elemento coincidente, retornara null.

```
var selectorVarios = document.querySelector(".contenedor");
```

Obtenemos como resultado:

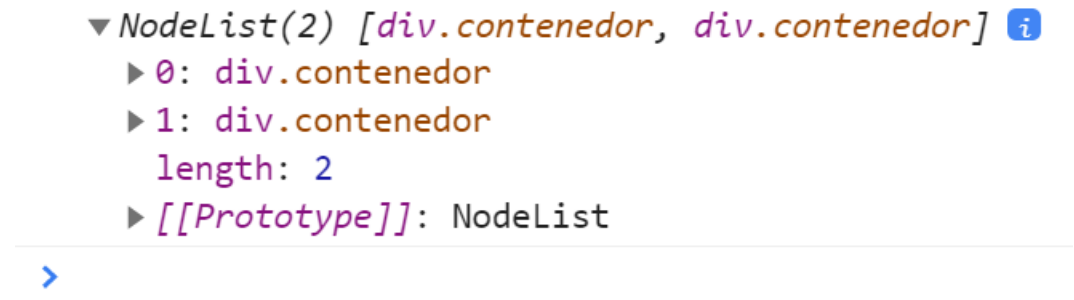


Para este caso, debemos especificar el símbolo del selector, similar como ocurre con **CSS**, por clase será (".clase"), ID ("#id") y por etiqueta ("h1").

- **Document.querySelectorAll**: devuelve una lista de cualquier elemento del **DOM** que coincida con el argumento que le indiquemos.

```
var selectorTodos = document.querySelectorAll(".contenedor");
```

Obtenemos como resultado:



Para obtener el acceso al contenido de un elemento, podemos hacer uso de la propiedad **innerHTML** a través del acceso de un objeto.

```
var selectorId = document.getElementById('parrafo').innerHTML;
```

Obteniendo como resultado:

```
Hola soy un párrafo.
>
```

En los casos en que hayamos obtenido una lista, la recorreremos con un ciclo:

```
var selectorClass = document.getElementsByClassName("contenedor");

for(let i = 0; i < selectorClass.length; i ++){
    console.log(selectorClass[i].innerHTML);
}
```

Obtendremos como resultado:

```
<p id="parrafo">Hola soy un párrafo.</p>

<h1>Soy un titulo</h1>

>
```

EXERCISE 2: MANIPULANDO LOS VALORES DEL DOM

Además de lo visto, podemos crear y eliminar elementos directamente desde **JavaScript**.

Vamos a observar, a través de un ejemplo, cómo podemos crear un nodo o, en otras palabras, crearemos una etiqueta desde nuestro **JavaScript** que insertaremos en el **HTML**.

Comenzaremos utilizando la propiedad `document.createElement`:

Esta propiedad crea un elemento **HTML** especificado por su *tagName*. Es importante destacar que este elemento solo se crea en memoria.

Normalmente se asigna a una variable, pero no pertenece al **DOM**, no forma parte de los nodos hasta que lo indiquemos posterior a su creación.

Crearemos un elemento `<p>`, colocando el *tagName* entre comillas en el paréntesis.

```
1 var creandoElemento = document.createElement("p");
```

Posterior a eso, indicaremos el texto que la etiqueta llevará, utilizando la propiedad `textContent`:

```
1 creandoElemento.textContent = "Este nodo fue creado desde JavaScript";
```

Finalmente, incorporamos el nodo al **HTML**, para eso, usaremos la propiedad `appendChild`.

Este método cumple la función de agregar un elemento al final de la lista de hijos de un elemento padre previamente especificado, es decir, agrega un elemento, pero al final de la lista completa de hijos que ese padre tenga.

Para entenderlo mejor, cada elemento dentro del **DOM** tiene un padre, por tanto, es hijo de algo. `<html>` es el padre de todos y tiene como hijos a `<head>` y `<body>`, que, a su vez, tiene como hijos todos los nodos creados en él, por ejemplo, en el `<head>` un hijo es el `<tittle>`.

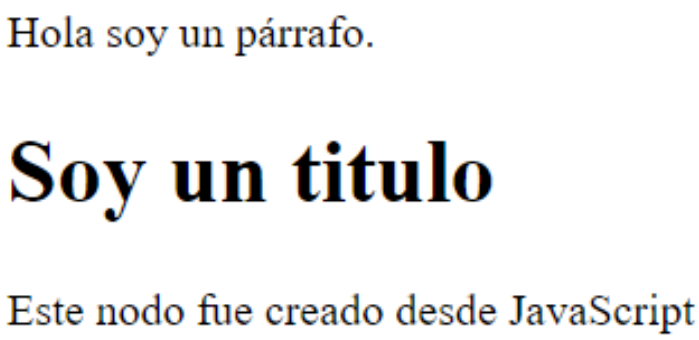
Para el caso de nuestro ejemplo, incorporaremos el nodo en el `<body>`, por lo que escribiremos:

```
1 document.body.appendChild(creandoElemento);
```

Quedando el código completo como:

```
1 var creandoElemento = document.createElement("p");
2 creandoElemento.textContent = "Este nodo fue creado desde JavaScript";
3 document.body.appendChild(creandoElemento);
```

Y obteniendo como resultado en el navegador:

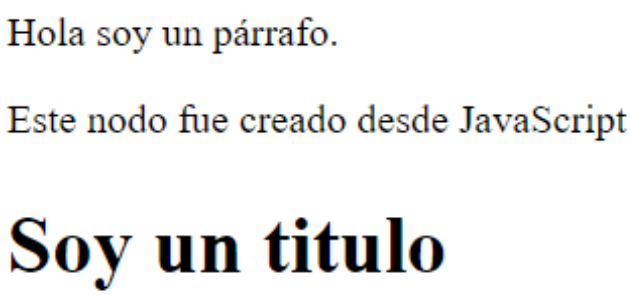


Podemos crear un Nodo e incorporarlo al **DOM** utilizando la propiedad `element.append`, la cual nos permite agregar más de un nodo.

Modifiquemos nuestro ejemplo. Utilizaremos **querySelector** para seleccionar una **class** y dentro de ese elemento, como su hijo, incorporaremos la misma **<p>** creada previamente.

```
var selectorVarios = document.querySelector(".contenedor");
var creandoElemento = document.createElement("p");
creandoElemento.textContent = "Este nodo fue creado desde JavaScript";
selectorVarios.append(creandoElemento);
```

Obteniendo como resultado:



Soy un titulo

Para eliminar un Nodo podemos hacer uso de la propiedad **removeChild**. Este método cumple la función de eliminar el Nodo hijo, siempre siendo hijo directo del Nodo al cual se hace referencia.

Eliminaremos la etiqueta **<p>** con **id** “párrafo”, hijo del ** ** con clase “contenedor”.

```
var nodoPadre = document.querySelector(".contenedor");
var nodoHijo = document.querySelector("#parrafo");

nodoPadre.removeChild(nodoHijo);
```

Nuestro resultado es:

EXERCISE 3: CAPTURANDO Y MODIFICANDO UN VALOR

Continuando con la manipulación del **DOM**, ya conocimos los selectores básicos y creamos y eliminamos nodos.

Mantenemos la estructura de nuestro **HTML,** capturaremos el valor de un elemento y lo modificaremos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-
scale=1.0">
  <title>Document</title>
</head>
<body>



   <div class="contenedor">
      <p id="parrafo">Hola soy un párrafo.</p>
   </div>

   <div class="contenedor">
      <h1 >Soy un título</h1>
   </div>

<script src="index.js"></script>
</body>
</html>
```

En **JavaScript** vamos a capturar el dato correspondiente al id “párrafo”, declarando una variable para almacenarlo y dándole un nuevo valor de inmediato, con el uso de la propiedad **innerHTML**.

```
var texto = document.getElementById("parrafo").innerHTML="Nuevo texto";
```

Provocando que, al momento de abrir nuestro archivo en el navegador o actualizarlo, el texto se encuentre modificado:

En alternativa podemos utilizar el elemento **innerText**.

```
var texto = document.getElementById("parrafo").innerText="Nuevo texto";
```

Obteniendo el mismo resultado del caso anterior.

Podemos observar que quizás no es muy práctico realizar una modificación de un elemento de esta forma, pero para solucionar aquello y realizar la modificación de elementos posterior a la interacción del usuario con nuestra página web, trabajaremos con eventos.

EXERCISE 4: CONOCIENDO LOS EVENTOS DE JAVASCRIPT

Cada una de las etiquetas **HTML** tiene su propia lista de posibles eventos que se le pueden asignar, sin embargo, conoceremos una lista con los eventos más importantes en **JavaScript**:

Observemos la siguiente tabla:

Evento	Descripción	Elementos para los que está definido
`onblur`	Un elemento pierde el foco	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Un elemento ha sido modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pulsar y soltar el ratón	Todos los elementos
`ondblclick`	Pulsar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Un elemento obtiene el foco	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onkeydown`	Pulsar una tecla y no soltarla	Elementos de formulario y `<body>`
`onkeypress`	Pulsar una tecla	Elementos de formulario y `<body>`
`onkeyup`	Soltar una tecla pulsada	Elementos de formulario y `<body>`
`onload`	Página cargada completamente	`<body>`
`onmousedown`	Pulsar un botón del ratón y no soltarlo	Todos los elementos
`onmousemove`	Mover el ratón	Todos los elementos
`onmouseout`	El ratón "sale" del elemento	Todos los elementos
`onmouseover`	El ratón "entra" en el elemento	Todos los elementos
`onmouseup`	Soltar el botón del ratón	Todos los elementos
`onreset`	Inicializar el formulario	`<form>`
`onresize`	Modificar el tamaño de la ventana	`<body>`
`onselect`	Seleccionar un texto	`<input>`, `<textarea>`
`onsubmit`	Enviar el formulario	`<form>`
`onunload`	Se abandona la página, por ejemplo al cerrar el navegador	`<body>`

Para ejemplificarlo mejor, vamos a crear un proyecto con un archivo **HTML** en el cual incorporaremos **Bootstrap** (solamente usaremos estilos) y lo enlazaremos con un archivo **JavaScript**.

En el **HTML** tendremos dos ** ** id “caja1” y “caja2” respectivamente. Dentro tendremos etiquetas **<p>** con algún texto.

Incorporaremos, desde **Bootstrap**, dos botones y finalmente un tercer ** ** con texto en **<h2>.**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">

  <title>Eventos</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootst
rap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
 crossorigin="anonymous">


  <script src="index.js"></script>
</head>
<body>

    <div id="caja1" style="padding: 10px; background-color:
chartreuse;" >
        <p>Texto que se ve desde el inicio</p>
    </div>

    <div id="caja2" style="padding: 10px; background-color: blue;
display: none;">
        <p>Texto oculto</p>
    </div>


    <div class="container my-5">
        <div class="btn btn-primary"  id="boton1">accion 1</div>
        <div class="btn btn-secondary"  id="boton2">accion 2</div>

```

```
    </div>

    <div style="background-color: coral;"  id="contenido">
        <h2 id="texto">Lorem, ipsum dolor sit amet consectetur
adipisicing elit. Beatae unde accusamus quibusdam, maxime sint, a
rem autem veniam quod sunt explicabo eligendi maiores porro
tempore tenetur quas incidunt laboriosam? Minus.</h2>
    </div>
</body>
</html>
```

Hemos incorporado algo de estilo en los ** ** y podemos observar que todos tienen su id, incluido el **<h2>.**

Ahora, incorporaremos nuestro primer evento. Podemos observar que el ** ** "caja2" lo hemos dejado oculto, utilizando la propiedad **"display: none"** de **CSS**. Lo que haremos, será que al momento de pasar el cursor del ratón por sobre el primer ** ** "caja1" se mostrará el mensaje del segundo .

Para eso, vamos a incorporar al primer ** ** el evento **onmouseover**, que, como se explica en nuestra tabla, activa alguna acción cuando el mouse "entra" al elemento.

Para activar una acción, vamos a hacer uso de una función que indique que acción realizar. La llamaremos **mostrarMensaje()**.

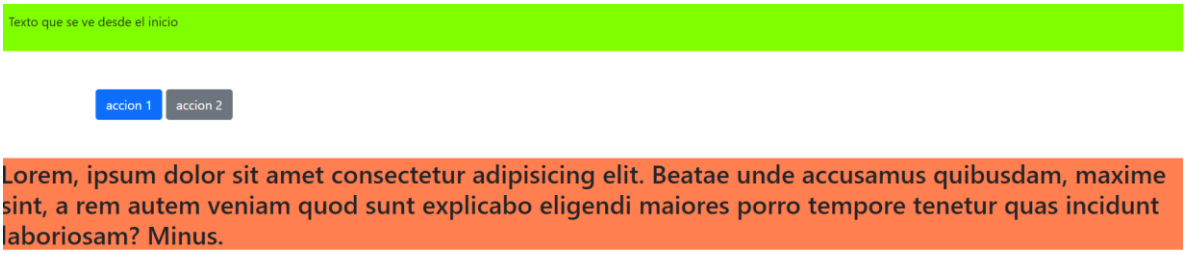
Nuestro ** ** quedará de la siguiente forma:

```
  <div id="caja1" style="padding: 10px; background-color: chartreuse;"
onmouseover="mostrarMensaje()" >
        <p>Texto que se ve desde el inicio</p>
    </div>
```

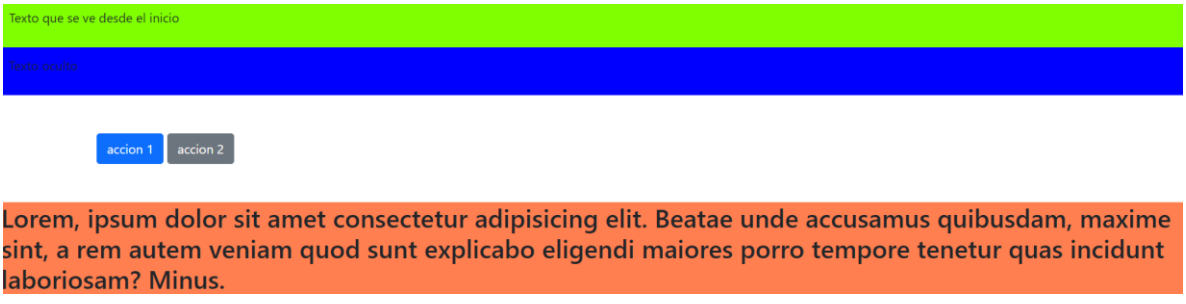
Ahora, nos dirigimos a nuestro archivo **JavaScript** y en él creamos la función **mostrarMensaje()**, con la cual indicaremos que el elemento de id “caja2” será mostrado, esto cambiando el **display** a **block** (lo contrario a **none**).

```
function mostrarMensaje(){
  document.getElementById('caja2').style.display = "block";
}
```

Nos dirigimos al navegador y observamos que nuestra página inicial se ve así:



Y al pasar el mouse por sobre nuestro ** ** ahora se verá así:



A esta altura, podrás notar que el problema es que, si dejamos un elemento oculto, probablemente es porque no queremos que se vea en todo momento y ahora, después de pasar por encima del elemento “caja1” el elemento “caja2” se muestra en todo momento.

Esto lo solucionaremos incorporando un nuevo evento al ** ** "caja1". Para esta ocasión incorporaremos el evento **onmouseout**, que se activa cuando el ratón sale del elemento.

Para indicar la acción a realizar, vamos a crear una nueva función que se llamará **ocultarMensaje()**, quedando de la siguiente manera.

```
1   <div id="caja1" style="padding: 10px; background-color: chartreuse;"
2 onmouseover="mostrarMensaje()" onmouseout="ocultarMensaje()">
3         <p>Texto que se ve desde el inicio</p>
4     </div>
```

En nuestro **JavaScript** vamos a crear la nueva función a la que le indicaremos **"display = none".**

```
1 function ocultarMensaje(){
2
3   document.getElementById('caja2').style.display = "none";
4 }
```

Actualizamos y podemos observar que ahora el mensaje del ** ** "caja2" se muestra u oculta dependiendo de la acción del mouse sobre el ** ** "caja1".

Ahora que tenemos listo este evento, vamos a continuar con los botones. A estos botones les daremos la posibilidad de modificar el texto que aparece justo abajo.

A ambos botones le colocaremos el evento **onclick**. Este nos permite activar el evento al hacer clic sobre un elemento.

Crearemos dos funciones, una para cada botón. El primero será **cambiarFondo()** y lo usaremos para cambiar el **background** del ** ** "contenido". El segundo será **cambiarTexto()**, cuya función será modificar el texto que se encuentra en el **<h2>** "texto".

De esta forma, los botones quedarán así:

```
    <div class="container my-5">
        <div class="btn btn-primary" onclick="cambiarFondo()"
id="boton1">accion 1</div>
        <div class="btn btn-secondary" onclick="cambiarTexto()"
id="boton2">accion 2</div>

    </div>
```

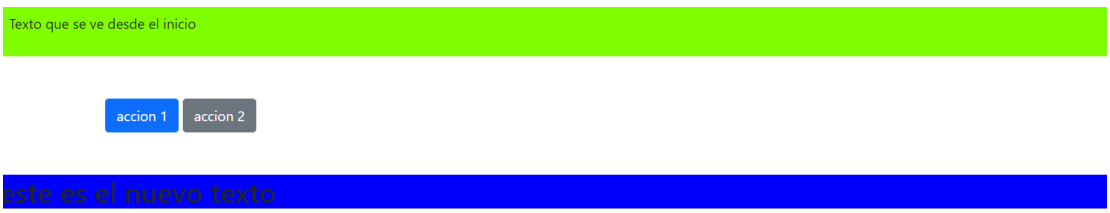
Vamos a nuestro **JavaScript** y en él crearemos las funciones necesarias para realizar lo que deseamos.

Utilizaremos, para el método **cambiarFondo()** **backgroundColor**. Y para **cambiarTexto()** usaremos **innerHTML**.

```
function cambiarFondo(){
  document.getElementById("contenido").style.backgroundColor = "blue";
}

function cambiarTexto(){
  document.getElementById("texto").innerHTML = "este es el nuevo
texto";
}
```

Ahora, podemos ver las modificaciones que realizamos:



Finalmente, vamos a realizar un último evento, en el primer botón: incorporaremos el evento **ondbclick**. Este método se activa con un doble clic sobre el elemento. Al realizar el doble clic llamaremos al método **volver()** y vamos a crearlo en nuestro **JavaScript**. Este nos permitirá volver a colocar el color coral.

Nuestro **HTML** completo quedará así:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">

  <title>Eventos</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap
.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">

  <script src="index.js"></script>
</head>
<body>
    <div id="caja1" style="padding: 10px; background-color:
chartreuse;" onmouseover="mostrarMensaje()"
onmouseout="ocultarMensaje()">
        <p>Texto que se ve desde el inicio</p>
    </div>

    <div id="caja2" style="padding: 10px; background-color: blue;
display: none;">
        <p>Texto oculto</p>
    </div>


    <div class="container my-5">
        <div class="btn btn-primary" onclick="cambiarFondo()"
ondblclick="volver()" id="boton1">accion 1</div>
```

```
33          <div class="btn btn-secondary" onclick="cambiarTexto()"
34 id="boton2">accion 2</div>
35      </div>
36      <div style="background-color: coral;"  id="contenido">
37          <h2 id="texto">Lorem, ipsum dolor sit amet consectetur
38 adipisicing elit. Beatae unde accusamus quibusdam, maxime sint, a rem
39 autem veniam quod sunt explicabo eligendi maiores porro tempore
40 tenetur quas incidunt laboriosam? Minus.</h2>
41      </div>
42 </body>
43 </html>
```

Y nuestro **JavaScript** quedará así:

```
 1 function mostrarMensaje(){
 2   document.getElementById('caja2').style.display = "block";
 3 }
 4 function ocultarMensaje(){
 5   document.getElementById('caja2').style.display = "none";
 6 }
 7 function cambiarFondo(){
 8   document.getElementById("contenido").style.backgroundColor = "blue";
 9 }
10 function cambiarTexto(){
11   document.getElementById("texto").innerHTML = "este es el nuevo
12 texto";
13 }
14 function volver(){
15   document.getElementById("contenido").style.backgroundColor =
16 "coral";
17 }
```

Ahora tenemos varios eventos activados trabajados con **JavaScript**.

Podemos leer más sobre eventos developer.mozilla.org